Variability and Complexity in Software Design – Towards a Research Agenda

Matthias Galster
University of Canterbury
Christchurch, New Zealand
mgalster@ieee.org

Danny Weyns
KU Leuven, Belgium
Linnaeus University, Sweden
danny.weyns@kuleuven.be

Michael Goedicke
University of Duisburg-Essen
Essen, Germany
michael.goedicke@s3.uni-due.de

Uwe Zdun
University of Vienna
Austria, Vienna
uwe.zdun@univie.ac.at

Rick Rabiser
Johannes Kepler University Linz
Linz, Austria
rick.rabiser@jku.at

Gilles Perrouin
University of Namur
Namur, Belgium
gilles.perrouin@unamur.be

Bo Zhang
Fraunhofer IESE
Kaiserslautern, Germany
bo.zhang@iese.fraunhofer.de

DOI: 10.1145/3011286.3011291
http://doi.acm.org/10.1145/3011286.3011291

ABSTRACT
Many of today's software systems accommodate different usage and deployment scenarios. Intentional and unintentional variability in functionality or quality attributes (e.g., performance) of software significantly increases the complexity of the problem and design space of those systems. The complexity caused by variability becomes increasingly difficult to handle due to the increasing size of software systems, new and emerging application domains, dynamic operating conditions under which software systems have to operate, fast moving and highly competitive markets, and more powerful and versatile hardware. This paper reports results of the first International Workshop on Variability and Complexity in Software Design that brought together researchers and engineers interested in the topic of complexity and variability. It also outlines directions the field might move in the future.

Categories and Subject Descriptors
D.2.11 [**Software Engineering**]: Software Architectures

General Terms
Management, Documentation, Design.

Keywords
Variability, complexity, software design.

1. INTRODUCTION
Today's software users expect flexibility from software in many dimensions, e.g., features, location and resource awareness, fault tolerance, energy consumption, etc. Therefore, many of today's software systems must accommodate different deployment and usage scenarios (e.g., product lines and families, self-adaptive systems, configurable or customizable single systems, open platforms, context-aware mobile apps, plug-ins of web browsers, service-based and cloud-based systems, Internet of Things, cyber-physical systems). These systems can range from small-scale embedded systems to large-scale enterprise software systems to ultra-large systems of systems. Variability can be intentional or unintentional and driven by many forces, for example, variations in users and user needs, dynamics in the availability of resources or external services, market segments, customer profiles, different emphases in different phases of the software development process, or variation in hardware resources. Therefore, variability needs to be addressed in a broader software engineering context and is not limited to "traditional" software product lines, the field in which variability has been discussed the most so far.

Intentional and unintentional variability in functionality and/or quality attributes of software significantly contributes to the complexity of the problem and design space of those systems. A design space comprises the set of possible design options and design parameters that could potentially meet a specific software system's requirements. Given the increasing size and heterogeneity of software systems (e.g., software ecosystems, cyber-physical systems, systems of systems, ultra-large scale systems), new and emerging application domains (e.g., unmanned aerial vehicles, smart health applications, large-scale surveillance systems, software-defined networking, social networking apps), dynamic operating conditions (e.g., availability of resources, variations in service availability, changing goals), fast moving and highly competitive markets (e.g., gaming, mobile apps), and increasingly powerful and versatile hardware (e.g., Raspberry Pi), the complexity caused by variability becomes more difficult to handle.

In some consumer domains of critical systems, e.g., autonomous and unmanned aerial vehicles (UAV), research is only slowly catching up with industry trends and needs [2]. Such systems can soon become an integral part of many industries, including construction, agriculture, emergency responder support, etc. Once this happens, practices need to be in place to help software engineers develop such systems. A particularly complex aspect of the engineering of such systems is the provision of quality assurances with sufficient confidence. Furthermore, successful companies are innovative companies that target new market opportunities, independent of solutions or ideas that currently exist. On the other hand, the time to market can make the difference between product success and failure. This highlights the need for "light-weight" approaches to variability-intensive systems, which balance the need for innovation but also consider reducing development effort, even for innovative products. New development models for variability-intensive systems could help manage system growth over time and offer opportunities for innovation throughout development. Also, there is a need-supply gap in engineering capability (processes, practices, skills

and workforce). Variability-intensive systems development differs from conventional software engineering in that conventional engineering does not address specifics of these systems, e.g., highly diverse stakeholders, extremely large design spaces, consistency checking amongst configurations/design options, etc. As mentioned in an ICSE Future of Software Engineering talk in 2014 [10], a trend/challenge in the next decade will be managing variability in a non-product line context and under open-world assumptions.

2. VACE WORKSHOP

2.1 History

The first edition of the International Workshop on Variability and Complexity in Software Design (VACE) was collocated with the International Conference on Software Engineering (ICSE 2016) in Austin, Texas. The workshop website can be found at http://vaquita-workshop.org/vace/. VACE is an evolution of the VARSA workshop series (International Workshop on Variability in Software Architecture) held at WICSA in 2011 [5], 2012 [7], and 2014 [6], and VAQUITA (Workshop on Variability for Qualities in Software Architecture) held at ECSA 2015. Evolving these two workshops into one ICSE workshop broadened the community beyond software architecture to reach an audience with a much broader and diverse background and expertise.

2.2 Overview

Variability has previously been targeted by various, separate software engineering sub-communities (e.g., requirements engineering, software architecture, product lines/families, service-orientation, self-adaptation), which should cooperate closer [8, 10]. Therefore, one of the key goals of this workshop was to provide one venue for researchers, practitioners and educators from different areas of software engineering to jointly discuss experiences, synergies, forge new collaborations, and explore innovative solutions that address the challenges of engineering for variability in high-quality software.

Designing for, implementing and maintaining variability in software systems not only affects characteristics of the software product and variability in functionality and quality (i.e., *what* do we build), e.g., systems with support for "continuous configuration management" from compile time and deployment time to runtime. It also affects the development process (i.e., *how* we build it), e.g., systematic quality assurance and validation despite a potentially large and highly complex design and solution space. Therefore, topics of interest for VACE included topics about product and process and how these interline with current trends in software development. This included software engineering issues related to requirements, design, implementation, evaluation, deployment, runtime adaptation, and maintenance of variability-intensive systems.

The workshop accepted different types of paper submissions (full papers, position and vision papers, industrial and empirical papers, education and training papers). Each submission was peer reviewed by three members of the program committee. Accepted papers were presented at the workshop and included in the proceedings (published by ACM). Around 20 participants attended the workshop. The workshop started with a keynote delivered by Christian Kästner from Carnegie Mellon University on quality assurance for highly-configurable software systems. Paper presentations focused on timely aspects of variability and complexity in software design and were organized in several sessions: variability at runtime; variability in practice; and domain-specific variability.

2.3 Keynote

Highly configurable systems can be tailored to specific use cases. When planned as software product lines, they can achieve orders of magnitude improvements in development costs, speed and quality compared to developing products one by one. At the same time, configuration options challenge quality assurance. Traditional analysis techniques, including type checking, static analysis and testing can analyze only one specific configuration at a time in an exponentially exploding configuration space. Dr. Christian Kästner provided an overview of work on variability-aware analysis that aims at analyzing all configurations of a configurable system in a single run, while exploiting the similarities between the configurations. In large design spaces, it is simply not possible to check each and every individual configuration. Analyzing configuration spaces as discussed in the talk goes beyond the context of software product lines since it applies to different types of variability-intensive systems, where analysis takes place at code level and includes compile-time variability. If features are defined in terms of #ifdef statements, then Linux for example has around 10,000 different features. Christian gave an overview of the TypeChef infrastructure that is able to parse and type-check C code with #ifdef variability, targeted at finding bugs in highly configurable systems such as the Linux kernel. Interestingly, there is almost no code in Linux that is shared by all configurations. In the second part of the talk, Christian went beyond ifdef's and discussed analyzing features at runtime (e.g., in Android, all code would be included for all possible configurations). Feature interactions at runtime could help to identify patterns and anti-patterns of interactions, thus identifying problematic code. Such information is useful when debugging and during code reviews. Christian also discussed the Varex infrastructure that pushes the idea of analyzing configurations toward testing.

3. OPEN RESEARCH TOPICS

Several open research topics were discussed during the workshop. In the long term we should try to organize these topics in a "whitelist" and a "blacklist": The whitelist contains topics that should be the focus of future research efforts. The blacklist on the other hand contains topics that a) have been addressed sufficiently in previous research or b) describe problems that will never be solved and therefore researchers and practitioners have to accept to live with them (unless there are strong arguments to (re-)open a debate). The following topics are discussed in more detail below:

- Lean processes and agile practices
- Continuous delivery/deployment and DevOps
- Impact of technology advances
- Variability in context
- Value-based variability
- Correctness of configurations
- Functional and quality variability
- Variability realization mechanisms
- Training and tools

3.1 Lean Processes and Agile Practices

Complex design spaces are particularly challenging for agile and lean processes. Flexible and lightweight approaches are needed to support variability in problem and solution space and to develop large-scale variability-intensive software. Industrial practice tends towards flexible and lightweight approaches [9]. On the other hand, variability requires anticipating design solutions for different usage and deployment scenarios. We need to understand whether there is a conflict between flexibility (agile/lean) and the need for bigger up-front design and design space exploration. This also includes challenges to balance business value and effort spent on anticipating variability.

3.2 Continuous Delivery/Deployment and DevOps

Today's systems are complex and often data-driven and organizations need to transform to support rapid continuous software production and delivery [4]. Therefore, we need design solutions to enable continuous delivery of variability-intensive systems. Furthermore, DevOps is becoming a trend in large systems development and deployment. We need to understand how DevOps could be implemented for development,

verification, deployment and maintenance of variability-intensive systems.

3.3 Impact of Technology Advances

New development technologies and frameworks are constantly appearing and evolving. Modern architectural approaches and technologies (e.g., microservices, containerization, nanoservices and "serverless" architectures, edge-cloud computing) may help us handle variability. However, new technologies also lead to new challenges for variability modelling. Particularly, modern architectures often follow a dynamic approach that supports the dynamic re-configuration and adaptation of systems during operation, e.g., as in cloud-based systems and cyber-physical systems. New variants might be introduced in such architectures at any time requiring support for the runtime co-evolution of variability models and systems. Consequently, variability models increasingly have to become "living entities" in such a context as frequently discussed in work on dynamic software product lines [11] and continuous deployment. Furthermore, technologies such as cloud computing and microservices might be drivers for trends like continuous deployment/delivery and DevOps (see previous section).

3.4 Variability in Context

Context describes circumstances that form the setting for an event, statement, or idea. We investigate context of variability in terms of (a) intentional and unintentional variability, and (b) emerging and maturing application and technology domains:

(a) *Intentional versus unintentional variability:* Intentional variability can be due to different customer profiles or usage scenarios, i.e., when variability offers an advantage. Therefore, intentional variability can also be a business strategy. Unintentional variability can be due to the effects of intentional variability or due to the effects of choosing different design solutions, i.e., when variability is not a goal but a side effect of other forces. We need approaches to limit unintentional variability, and ways to better scope intentional variability to manage complexity.

(b) *Emerging and maturing application and technology domains:* Variability in emerging and maturing domains, and in particular in end-user domains, e.g., big data, UAV and software-defined networking (SDN) impose new challenges due to potentially highly diverse application and uncertain deployment scenarios and thus more possible variability. Also, emerging domains include domains that are subject to regulations and legal aspects, e.g., regulated domains such as aviation. These not only affect the software part of critical systems but also hardware (e.g., sensors, actuators and controllers in wearable computing applications in the medical domain), which makes the exploration of design spaces of variability-intensive systems and their verification and validation even more challenging. This is particularly the case when uncertainties (e.g., the actual deployment conditions) may only be resolvable at runtime.

3.5 Value-based Variability

Today's highly customizable variability-intensive systems offer an extremely high degree of technical variability, intentionally as well as unintentionally. However, not all technically possible variants of a system are also relevant and meaningful for system users. While product line scoping approaches offer some guidance, in practice modelers still struggle to find the right balance between what variability *could* be modeled and what variability *should* be modeled.

Linking business issues with technology issues has received increased attention in software engineering. For example, the field of value-based software engineering (VBSE) aims to overcome the traditional value-neutral approach in software engineering that treats all artifacts as equally important. Value-based variability modeling and management [12] considers the business value and the associated risks of variability during modeling, and not only when defining the scope of a product line. Furthermore, VBSE suggests that variability management must not be seen as a pure modeling problem. Extracting tacit variability knowledge from diverse heterogeneous stakeholders is a collaborative process that relies on involving software engineers that have been designing and developing the reusable assets as well as people marketing and selling these assets.

3.6 Correctness of Configurations

The ability of variability-aware software to produce correct solutions will ultimately determine its success. The impact of incorrect configurations can range from the display of a wrong price while buying goods using an online configurator to OS kernels that cannot be compiled. Given the combinatorial explosion of the number of configurations induced by variability, guaranteeing correctness is a challenging task.

Two kinds of strategies can be thought of:

- At the domain engineering level, compact notations such *as featured transition systems* [3]enable verification of the whole configuration space, ensuring that abstract configurations cannot violate a given set of properties. This nevertheless requires a fully identifiable configuration space and a relatively abstract way of handling configurations behavior to keep the analysis traceable.
- The second kind of strategies takes advantage of the application engineering process to perform analyses while the configuration is under construction. These strategies may be able to cope with an unknown configuration space (e.g. self-adaptive architectures, where analysis may be partially performed at runtime [13]) and perform more fine-grained analysis at the product level. However, product-by-product verification limits reuse opportunities.

A combination of these two strategies can be fruitful to leverage their mutual benefits and to mitigate their drawbacks. This combination will rely on flexible software architectures that allow different kinds of reasoning to co-operate efficiently and reduce overall complexity.

3.7 Functional and Quality Variability

In order to meet functional and quality requirements of variability-intensive systems, we may need specific design practices. For example, what are suitable models and mechanisms to handle variability, from inception to operation? Related topics include modeling of variability across different life-cycle stages of software systems; patterns, styles and tactics; practices for requirements engineering, architecting, design, implementation, testing and maintenance of variability-intensive systems, methods for quality assurance, process and product metrics for variability-intensive systems; and reference models, reference architectures, patterns and frameworks to reuse design knowledge when engineering with variability in mind.

3.8 Variability Realization Mechanisms

In the phases of variability design and realization, one of the most critical decisions is the selection of variability realization mechanisms, such as Cloning, Conditional Compilation, Conditional Execution, Polymorphism, Module Replacement, Runtime Reconfiguration, etc. Practical experiences show that there is not a single variability mechanism that is appropriate in every situation, but each of the mechanisms has its own pros and cons. Therefore, it is crucial, but often difficult, to decide which mechanism should be used in which situation (e.g., depending on code granularity, change frequency, binding time, etc.). To this end, a practical guideline including some cost-benefit estimation support for each mechanism would be very helpful.

Moreover, as a variability-aware system evolves over time, the context factors of an existing mechanism in use might become inappropriate, typically making the variability code overly complex and hard to maintain. It would be necessary to refactor the variability realizations with another mechanism. However, it is difficult to make decisions like when to conduct such refactoring using which new mechanism. Further research with empirical evidences remains to be done in this direction.

3.9 Training and Tools

Training refers to how to educate students and practitioners in the skills required when coping with issues discussed under the topics above. Teaching variability (modeling) skills is challenging as recently discussed based on the results of a survey [1]. For instance, not only the complexity of the subject – software engineering is already a complex subject, even without considering variability – and required background knowledge complicates teaching, there is also a lack of well-documented real-world examples and case studies suitable for teaching (as opposed to existing case studies for research).

Furthermore, supporting issues raised under the topics above should be seamlessly integrated with development processes. Therefore, we need tools that help analyze, design for, implement and maintain systems with variability in mind.

4. CONCLUSIONS

We summarized the outcome of the first International Workshop on Variability and Complexity in Software Design. We gave an overview of the event, summarized discussions and offered an outlook on themes that emerged from the discussions at the workshop and which might be subject to future work.

In addition to the research topics outlined above, we believe that it is important to focus on quality forums for researchers and practitioners to grow the community and keep it active and to foster cross-pollination between events. There are several community events related to variability, e.g., SPLC, VaMoS, ICSR, SEAMS, ICSME. Also, as discussed throughout this report, different software engineering areas and topics "grow together" and variability is a cross-cutting concern. Therefore, variability can also be a subject at other events, e.g., on topics related to fast-paced and continuous delivery/development in context and highly flexible environments, such as the RCoSE workshop series.

5. ACKNOWLEDGMENTS

The VACE workshop is a collective endeavor. The organizers would like to thank all workshop authors, presenters and submitters. We also thank the ICSE 2016 organizers and in particular the workshop chairs. Finally, we are grateful to the members of the program committee.

6. REFERENCES

[1] Acher, M., Lopez-Herrejon, R. E., and Rabiser, R., "A Survey on Teaching of Software Product Lines," in *8th International Workshop on Variablity Modelling of Software-intensive Systems (VaMoS)* Nice, France: ACM, 2014, pp. 3-10.

[2] Backstory 2015. Boom Times for the New Camera Boom. *IEEE Spectrum* 52, 7 (2015), 3.

[3] Classen, A., Cordy, M., Schobbens, P.-Y., Heymans, P., Legay, A., and Raskin, J. F. 2013. Featured Transition Systems: Foundations for Verifying Variability-Intensive Systems and Their Application to LTL Model Checking. *IEEE Transactions on Software Engineering* 39, 8 (2013), 1069-1089.

[4] Fitzgerald, B. and Stol, K.-J. 2015. Continous Software Engineering - a Roadmap and Agenda. *Journal of Systems and Software* in press, (2015),

[5] Galster, M., Avgeriou, P., Weyns, D., and Mannisto, T. 2011. Variability in Software Architecture: Current Practices and Challenges. *ACM SIGSOFT Software Engineering Notes* 36, 5 (2011), 30-32.

[6] Galster, M., Mannisto, T., Weyns, D., and Avgeriou, P. 2014. Variability in Software Architecture: The Road Ahead. *ACM SIGSOFT Software Engineering Notes* 39, 4 (2014), 33-34.

[7] Galster, M., Weyns, D., Avgeriou, P., and Becker, M. 2013. Variability in Software Architecture: Views and Beyond. *ACM SIGSOFT Software Engineering Notes* 38, 1 (2013), 46-49.

[8] Galster, M., Weyns, D., Tofan, D., Michalik, B., and Avgeriou, P. 2014. Variability in Software Systems - A Systematic Literature Review. *IEEE Transactions on Software Engineering* 40, 3 (2014), 282-306.

[9] Keeling, M. 2015. Lightweight and Flexible - Emerging Trends in Software Architecture from the SATURN Conferences. *IEEE Software* 32, 3 (2015), 7-11.

[10] Metzger, A. and Pohl, K., "Software Product Line Engineering and Variability Management: Achievements and Challenges," in *Future of Software Engineering* Hyderabad, India: ACM, 2014, pp. 70-84.

[11] Quinton, C., Rabiser, R., Vierhauser, M., Gruenbacher, P., and Baresi, L., "Evolution in Dynamic Software Product Lines: Challenges and Perspectives," in *19th International Conference on Software Product Lines* Nashville, TN: ACM, 2015, pp. 126-130.

[12] Rabiser, R., Dhungana, D., Gruenbacher, P., and Burgstaller, B., "Value-based Elicitation of Product Line Variability: An Experience Report," in *2nd International Workshop on Variability Modelling of Software-intensive Systems (VaMoS)* Essen, Germany: ICB Research, 2008, pp. 73-79.

[13] Weyns, D., Bencomo, N., Calinescu, R., Camara, J., Ghezzi, C., Grassi, V., Grunske, L., Inverardi, P., Jezequel, J.-M., Malek, S., Mirandola, R., Mori, M., and Tamburrelli, G. 2016 Perpetual Assurances for Self-Adaptive Systems. In *Software Engineering of Self-adaptive Systems*, R. de Lemos, D. Garlan, C. Ghezzi, and H. Giese, Eds. Springer, Berlin/Heidelberg.